CS634

DATA MINING PROJECT

vm567

Creating a USPTO application to calculate the Patentability score of different datasets.

United States Patent and Trademark Office application predicts a patentability score of a dataset based on the information provided in the dataset.

Source code :

```
from pprint import pprint

from datasets import load_dataset

from transformers import AutoTokenizer,pipeline

from torch.utils.data import DataLoader

import streamlit as st

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score

import torch

from transformers import TrainingArguments, Trainer

from transformers import BertTokenizer, BertForSequenceClassification,AutoTokenizer,AutoModelForSequenceClassification
```

```
dataset_dict = load_dataset('HUPD/hupd',
    name='sample',
 data_files="https://huggingface.co/datasets/HUPD/hupd/blob/main/hupd_metadata_2022-02-22.feather",
    icpr_label=None,
    train_filing_start_date='2016-01-01',
    train_filing_end_date='2016-01-21',
    val_filing_start_date='2016-01-22',
    val_filing_end_date='2016-01-31',
)
st.write("hello world")
print('Loading is done!')
print(dataset_dict)
print(f'Train dataset size: {dataset_dict["train"].shape}')
print(f'Validation dataset size: {dataset_dict["validation"].shape}')


decision_to_str = {'REJECTED': 0, 'ACCEPTED': 1, 'PENDING': 2, 'CONT-REJECTED': 3, 'CONT-ACCEPTED': 4, 'CONT-PENDING': 5}
def map_decision_to_string(example):
    return {'decision': decision_to_str[example['decision']]}
train_set = dataset_dict['train'].map(map_decision_to_string)
val_set = dataset_dict['validation'].map(map_decision_to_string)


train_df=train_set.data.to_pandas()
```

```
val_set = val_set.data.to_pandas()
train_df = train_df.drop(train_df[train_df['decision'] > 1 ].index)
val_set=val_set.drop(val_set[val_set['decision'] > 1].index)


train_df_req = train_df[['patent_number','abstract','claims','decision']]
val_df_req=val_set[['patent_number','abstract','claims','decision']]
option = st.selectbox(
    'How would you like to be contacted?',
    ('Email', 'Home phone', 'Mobile phone'))
X_train_col = train_df_req[['abstract','claims']]
Y_train_col = train_df_req['decision']
X_val_col = val_df_req[['abstract','claims']]
Y_val_col = val_df_req['decision']


print(X_train_col.head())
print(Y_train_col.head())
from transformers import DistilBertForSequenceClassification, DistilBertTokenizer,
  DistilBertConfig
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',num_labels=2)
X_train, X_test, y_train, y_test = train_test_split(X_train_col, Y_train_col, test_size=0.2)
print(X_train)
class Dataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels=None):
        self.encodings = encodings
        self.labels = labels
    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        if self.labels:
            item["labels"] = torch.tensor(self.labels[idx])
        return item
    def __len__(self):
        return len(self.encodings["input_ids"])
X_train_encodings = tokenizer(list(X_train),padding = True, truncation = True,max_length=512)
X_test_encodings = tokenizer(list(X_test),padding = True, truncation = True,max_length=512)
X_val_col_encodings = tokenizer(list(X_val_col),padding = True, truncation = True,max_length=512)
x_train_dataset = Dataset(X_train_encodings,y_train)
X_test_dataset = Dataset(X_test_encodings,y_test)
X_val_dataset = Dataset(X_val_col_encodings,Y_val_col)
def compute_metrics(p):
    print(type(p))
    pred, labels = p
    pred = np.argmax(pred, axis=1)
    accuracy = accuracy_score(y_true=labels, y_pred=pred)
    recall = recall_score(y_true=labels, y_pred=pred)
    precision = precision_score(y_true=labels, y_pred=pred)
    f1 = f1_score(y_true=labels, y_pred=pred)
    return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1": f1}
args = TrainingArguments(
    output_dir="output",
    num_train_epochs=1,
    per_device_train_batch_size=8
)
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=x_train_dataset,
    eval_dataset=X_test_dataset,
    compute_metrics=compute_metrics
)
trainer.train()
```

OUTPUT: